Summarizing aggregated data, Part 1

SQL Server's ROLLUP clause lets you put totals, averages and more from subsets of your computed dta right into the same cursor.

Tamar E. Granor, Ph.D.

This series of articles looks at features of SQL Server that make some tasks easier than they are with VFP's SQL sublanguage. Over the next two issues, we explore ways to provide summary results for all data and larger subgroups of your data as part of query results. In this issue, we look at ROLLUP.

SQL SELECT's GROUP BY clause makes it easy to aggregate data in a query. Just include the fields that specify the groups and some fields using the aggregate functions (COUNT, SUM, AVG, MIN, MAX in VFP; SQL Server has a few more).

For example, the query in Listing 1 (TotalsByCountryCity.PRG in this month's downloads) fills a cursor with sales for each city for each month; Figure 1 shows partial results.

Listing 1. This query computes total sales for each combination of country, city, year and month.

```
SELECT Country, ;
       City, ;
       YEAR(OrderDate) AS OrderYear, ;
       MONTH(OrderDate) AS OrderMonth, ;
       SUM(Quantity * OrderDetails.UnitPrice);
         AS nTotal ;
       AVG(Quantity * OrderDetails.UnitPrice);
         AS nAvg, ;
       COUNT(*) AS nCount ;
  FROM Customers ;
    JOIN Orders ;
      ON Customers.CustomerID = ;
         Orders.CustomerID ;
    JOIN OrderDetails ;
      ON Orders.OrderID = ;
         OrderDetails.OrderID ;
GROUP BY OrderYear, OrderMonth, ;
         Country, City ;
ORDER BY Country, City, ;
         OrderYear, OrderMonth ;
INTO CURSOR csrCtyTotals
```

You can do an analogous query using the SQL Server AdventureWorks 2008 database, though it involves a lot more tables because the AdventureWorks database covers a wider range of data than just sales. Listing 2 (SalesByCountryCity.SQL in this month's downloads) shows the analogous SQL Server query.

Listing 2. Aggregating the data with SQL Server's AdventureWorks 2008 database is more verbose, but contains the same elements.

```
SELECT Person.CountryRegion.Name,
       Person.Address.City,
       YEAR(OrderDate) AS nYear,
       MONTH(OrderDate) AS nMonth,
       SUM(SubTotal) AS TotalSales
       AVG(SubTotal) AS AvgSale,
       COUNT(*) AS NumSales
  FROM Sales.Customer
    JOIN Person.Person
      ON Customer.PersonID =
         Person.BusinessEntityID
    JOIN Person.BusinessEntityAddress
      ON Person.BusinessEntityID =
        BusinessEntityAddress.BusinessEntityID
```

Country	City	Orderyear	Ordermonth	Ntotal	Navg	Ncount
Argentina	Buenos Aires	1997	1	319.2000	159.6000	2
Argentina	Buenos Aires	1997	2	443.4000	221.7000	2
Argentina	Buenos Aires	1997	4	225.5000	75.1667	3
Argentina	Buenos Aires	1997	5	110.0000	110.0000	1
Argentina	Buenos Aires	1997	10	706.0000	235.3333	3
Argentina	Buenos Aires	1997	12	12.5000	12.5000	1
Argentina	Buenos Aires	1998	1	1409.0000	352.2500	4
Argentina	Buenos Aires	1998	2	866.7000	173.3400	5
Argentina	Buenos Aires	1998	3	3645.8000	405.0889	9
Argentina	Buenos Aires	1998	4	381.0000	95.2500	4
Austria	Graz	1996	7	4483.4000	640.4857	7
Austria	Graz	1996	11	7511.8000	938.9750	8
Austria	Graz	1996	12	5175.2000	575.0222	9
Austria	Graz	1997	1	9515.4000	1189.4250	8

Figure 1. The query in Listing 1 computes the total sales for each city in each month.

```
JOIN Person.Address
      ON BusinessEntityAddress.AddressID =
         Address.AddressID
    JOIN Person.StateProvince
      ON Address.StateProvinceID =
         StateProvince.StateProvinceID
    JOIN Person.CountryRegion
      ON StateProvince.CountryRegionCode =
         CountryRegion.CountryRegionCode
    JOIN Sales.SalesOrderHeader
      ON Customer.CustomerID =
         SalesOrderHeader.CustomerID
    JOIN Sales.SalesOrderDetail
      ON SalesOrderHeader.SalesOrderID =
         SalesOrderDetail.SalesOrderID
  GROUP BY CountryRegion.Name, Address.City,
           YEAR(OrderDate), MONTH(OrderDate))
```

The rules for grouping are pretty simple. The field list contains two types of fields, those to group on, and those that are being aggregated. In the VFP example, the fields to group on are Country, City, OrderYear and OrderMonth, and the aggregated fields are nTotal, nAvg and nCount. The SQL Server query has the same list, but the field names are different. (Before VFP 8, you could include fields in the list that were neither grouped on nor aggregated, but doing so could give you misleading results. This article on my website explains the problem in detail: http://tinyurl.com/leydyqw.)

Computing group totals

What the basic query doesn't give you, though, is aggregation (that is, summaries) at any level except the one you specify. That is, while you get the total, average and count for a specific city in a specific month, you don't get them for that city for the whole year, or for that month for a whole country, and so on. Figure 2 shows what we're looking for. At the end of each year, a new record shows the total, average and count for that year. At the end of each city, another record shows the city's total, average and count and at the end of each country, yet another record has country-wide results.

In VFP, there are three ways to get that data. One is to create a report and use totals and report variables to compute and report that data, but of course, then you only have the data as output, not in a VFP cursor.

The second choice is to use Xbase code to compute them based on the initial cursor. Listing 3 (WithGroupTotalsXbase.PRG in this month's downloads) shows how to do this; it assumes you've already run the query in Listing 1. It keeps running totals and counts for each level: year, city, country and overall. Then, when one of those changes, it inserts the appropriate record.

Listing 3. You can add subgroup aggregates by looping through the cursor.

```
LOCAL nYearTotal, nCityTotal, ;
      nCountryTotal, nGrandTotal
LOCAL nYearCnt, nCityCnt, ;
      nCountryCount, nGrandCount
LOCAL nCurYear, cCurCity, cCurCountry

* Create a new cursor to hold the results
SELECT * ;
  FROM csrCtyTotals ;
  WHERE .F. ;
  INTO CURSOR csrWithGroupTotals READWRITE

SELECT csrCtyTotals
STORE 0 TO nYearTotal, nCityTotal, ;
```

Country	City	Orderyear	Ordermonth	Ntotal	Navg	Ncount
Austria	.NULL.	.NULL.	.NULL.	139496.6300	877.3373	159
Belgium	Bruxelles	1997	5	946.0000	315.3333	3
Belgium	Bruxelles	1997	8	1434.0000	717.0000	2
Belgium	Bruxelles	1997	12	3304.0000	1101.3333	3
Belgium	Bruxelles	1997	.NULL.	5684.0000	710.5000	8
Belgium	Bruxelles	1998	2	2950.5000	983.5000	3
Belgium	Bruxelles	1998	3	1500.7000	375.1750	4
Belgium	Bruxelles	1998	4	295.3800	147.6900	2
Belgium	Bruxelles	1998	.NULL.	4746.5800	527.3978	9
Belgium	Bruxelles	.NULL.	.NULL.	10430.5800	613.5635	17
Belgium	Charleroi	1996	7	3730.0000	1243.3333	3
Belgium	Charleroi	1996	9	2708.8000	902.9333	3
Belgium	Charleroi	1996	.NULL.	6438.8000	1073.1333	6
Belgium	Charleroi	1997	2	3891.0000	778.2000	5
Belgium	Charleroi	1997	3	2484.1000	496.8200	5
Belgium	Charleroi	1997	12	28.0000	28.0000	1
Belgium	Charleroi	1997	.NULL.	6403.1000	582.1000	11
Belgium	Charleroi	1998	1	5693.0000	813.2857	7
Belgium	Charleroi	1998	2	1209.0000	302.2500	4
Belgium	Charleroi	1998	3	2455.0000	613.7500	4
Belgium	Charleroi	1998	4	2505.5000	357.9286	7
Belgium	Charleroi	1998	.NULL.	11862.5000	539.2045	22
Belgium	Charleroi	.NULL.	.NULL.	24704.4000	633.4462	39
Belgium	.NULL.	.NULL.	.NULL.	35134.9800	163.4185	215

Figure 2. It can be useful to have group totals in the same cursor as the original data.

```
          nCountryTotal, nGrandTotal
STORE 0 TO nYearCount, nCityCount, ;
          nCountryCount, nGrandCount
nCurYear = csrCtyTotals.OrderYear
cCurCity = csrCtyTotals.City
cCurCountry = csrCtyTotals.Country

SCAN
  * First check for end of year,
  * but could be same year and change of city
  * or country.
  IF csrCtyTotals.OrderYear <> m.nCurYear OR ;
     NOT (csrCtyTotals.City == m.cCurCity) OR;
     NOT (csrCtyTotals.Country == ;
          m.cCurCountry)
    INSERT INTO csrWithGroupTotals ;
      VALUES (m.cCurCountry, m.cCurCity, ;
              m.nCurYear, .null., ;
              m.nYearTotal, ;
              m.nYearTotal/m.nYearCount, ;
              m.nYearCount)
    m.nCurYear = csrCtyTotals.OrderYear
    STORE 0 TO m.nYearTotal, m.nYearCount

    * Now check for change of city
    IF NOT (csrCtyTotals.City == m.cCurCity) ;
       OR ;
       NOT (csrCtyTotals.Country == ;
            m.cCurCountry)
        INSERT INTO csrWithGroupTotals ;
            VALUES (m.cCurCountry, ;
                    m.cCurCity, ;
                    .null., .null., ;
                    m.nCityTotal, ;
                m.nCityTotal/m.nCityCount, ;
                    m.nCityCount)
          m.cCurCity = csrCtyTotals.City
          STORE 0 TO m.nCityTotal, ;
                     m.nCityCount

          * Now check for change of country
          IF NOT (csrCtyTotals.Country == ;
                  m.cCurCountry)
            INSERT INTO csrWithGroupTotals ;
              VALUES (m.cCurCountry, .null., ,
                      .null., .null., ;
                       m.nCountryTotal, ;
              m.nCountryTotal/m.nCountryCount, ;
                       m.nCountryCount)
            m.cCurCountry = ;
              csrCtyTotals.Country
            STORE 0 TO m.nCountryTotal, ;
                       m.CountryCount
          ENDIF
       ENDIF
  ENDIF

  * Now handle current record
  INSERT INTO csrWithGroupTotals ;
    VALUES (csrCtyTotals.Country, ;
            csrCtyTotals.City, ;
            csrCtyTotals.OrderYear, ;
            csrCtyTotals.OrderMonth, ;
            csrCtyTotals.nTotal, ;
            csrCtyTotals.nAvg, ;
            csrCtyTotals.nCount)
  nYearTotal = m.nYearTotal + ;
               csrCtyTotals.nTotal
  nYearCount = m.nYearCount + ;
               csrCtyTotals.nCount
  nCityTotal = m.nCityTotal + ;
               csrCtyTotals.nTotal
  nCityCount = m.nCityCount + ;
               csrCtyTotals.nCount
  nCountryTotal = m.nCountryTotal + ;
                  csrCtyTotals.nTotal
  nCountryCount = m.nCountryCount + ;
                  csrCtyTotals.nCount
  nGrandTotal = m.nGrandTotal + ;
                csrCtyTotals.nTotal
  nGrandCount = m.nGrandCount + ;
                csrCtyTotals.nCount

ENDSCAN

* Now insert grand totals
INSERT INTO csrWithGroupTotals ;
  VALUES (.null., .null., .null., .null., ;
          m.nGrandTotal, ;
          m.nGrandTotal/m.nGrandCount, ;
          m.nGrandCount)
```

The third choice is to do a series of queries, each grouping on different levels and then consolidate the results. Listing 4 shows this version of the code; as in the previous example, it assumes you've already run the query that creates csrCtyTotals. This code creates a cursor with each city's annual totals, one with each city's overall totals, one with each country's overall totals, and one containing the grand total. Then it uses UNION to combine all the results into a single cursor. It's included in this month's downloads as WithGroupTotalsSQL.PRG.

Listing 4. You can add the yearly, city-wide and country-wide totals using SQL, as well.

```
* Now year totals by city
SELECT Country, City, OrderYear, ;
       999 as OrderMonth, ;
       SUM(nTotal) AS nTotal, ;
       SUM(nTotal)/SUM(nCount) AS nAvg, ;
       SUM(nCount) AS nCount ;
  FROM csrCtyTotals ;
  GROUP BY Country, City, OrderYear ;
  INTO CURSOR csrYearTotals

* Now city totals by year
SELECT Country, City, ;
       99999 AS OrderYear, ;
       999 as OrderMonth, ;
       SUM(nTotal) AS nTotal, ;
       SUM(nTotal)/SUM(nCount) AS nAvg, ;
       SUM(nCount) AS nCount ;
  FROM csrCtyTotals ;
  GROUP BY Country, City ;
  INTO CURSOR csrCityTotals

* Now year totals
SELECT Country, ;
       REPLICATE('Z', 15) AS City, ;
       99999 AS OrderYear, ;
       999 as OrderMonth, ;
       SUM(nTotal) AS nTotal, ;
       SUM(nTotal)/SUM(nCount) AS nAvg, ;
       SUM(nCount) AS nCount ;
  FROM csrCtyTotals ;
  GROUP BY Country ;
  INTO CURSOR csrCountryTotals

* Now grand total
SELECT REPLICATE('Z', 15) AS Country, ;
       REPLICATE('Z', 15) AS City, ;
       99999 AS OrderYear, ;
       999 as OrderMonth, ;
       SUM(nTotal) AS nTotal, ;
       SUM(nTotal)/SUM(nCount) AS nAvg, ;
       SUM(nCount) AS nCount ;
```

```
  FROM csrCtyTotals ;
  INTO CURSOR csrGrandTotal

* Create one cursor
SELECT * ;
  FROM csrCtyTotals ;
UNION ALL ;
SELECT * ;
  FROM csrYearTotals ;
UNION ALL ;
SELECT * ;
  FROM csrCityTotals ;
UNION ALL ;
SELECT * ;
  FROM csrCountryTotals ;
UNION ALL ;
SELECT * ;
  FROM csrGrandTotal ;
  ORDER BY Country, City, ;
           OrderYear, OrderMonth ;
  INTO CURSOR csrWithGroupTotals READWRITE

UPDATE csrWithGroupTotals ;
  SET OrderMonth = .null. ;
  WHERE OrderMonth = 999

UPDATE csrWithGroupTotals ;
  SET OrderYear = .null. ;
  WHERE OrderYear = 99999

UPDATE csrWithGroupTotals ;
  SET City = .null. ;
  WHERE City = REPLICATE('Z', 15)

UPDATE csrWithGroupTotals ;
  SET Country = .null. ;
  WHERE Country = REPLICATE('Z',
15)
```

There's one trick in this code. If we put null into the fields that are irrelevant for a given total, when we sort the result, the totals appear above rather than below the records they represent. Instead, we put an impossible value that sorts to the bottom initially, then change it to null after ordering the data.

Introducing ROLLUP

Of course, the reason for showing all this code is that SQL Server makes it much easier. The ROLLUP clause lets you compute these summaries as part of the original query.

ROLLUP appears in the GROUP BY clause, looking like a function around the fields you want to apply it to. Listing 5 shows the SQL Server equivalent of Listing 3 and Listing 4; the code is included in this month's downloads as SalesByCountryCityRollup.SQL. Figure 3 shows partial results.

Listing 5. SQL Server's ROLLUP clause computes the subgroup aggregates as part of the query.

```
SELECT Person.CountryRegion.Name,
       Person.Address.City,
       YEAR(OrderDate) AS nYear,
       MONTH(OrderDate) AS nMonth,
       SUM(SubTotal) AS TotalSales,
       AVG(SubTotal) AS AvgSale,
       COUNT(SubTotal) AS NumSales
FROM Sales.Customer
  JOIN Person.Person
    ON Customer.PersonID =
       Person.BusinessEntityID
  JOIN Person.BusinessEntityAddress
    ON Person.BusinessEntityID =
      BusinessEntityAddress.BusinessEntityID
  JOIN Person.Address
    ON BusinessEntityAddress.AddressID =
       Address.AddressID
  JOIN Person.StateProvince
    ON Address.StateProvinceID =
       StateProvince.StateProvinceID
  JOIN Person.CountryRegion
    ON StateProvince.CountryRegionCode =
       CountryRegion.CountryRegionCode
  JOIN Sales.SalesOrderHeader
    ON Customer.CustomerID =
       SalesOrderHeader.CustomerID
  JOIN Sales.SalesOrderDetail
    ON SalesOrderHeader.SalesOrderID =
       SalesOrderDetail.SalesOrderID
GROUP BY ROLLUP(CountryRegion.Name,
                Address.City,
                YEAR(OrderDate),
                MONTH(OrderDate))
```

Name	City	nYear	nMonth	TotalSales	AvgSale	NumSales
Australia	Caloundra	2007	NULL	203478.1...	1225.7718	166
Australia	Caloundra	2008	1	19240.68	962.034	20
Australia	Caloundra	2008	2	31246.21	801.1848	39
Australia	Caloundra	2008	3	50989.52	1456.8434	35
Australia	Caloundra	2008	4	50096.48	1192.7733	42
Australia	Caloundra	2008	5	34395.28	1433.1366	24
Australia	Caloundra	2008	6	25243.12	901.54	28
Australia	Caloundra	2008	7	494.28	61.785	8
Australia	Caloundra	2008	NULL	211705.57	1080.1304	196
Australia	Caloundra	NU...	NULL	527130.8...	1301.5576	405
Australia	Cloverd...	2005	8	3399.99	3399.99	1
Australia	Cloverd...	2005	10	3374.99	3374.99	1
Australia	Cloverd...	2005	11	3578.27	3578.27	1
Australia	Cloverd...	2005	12	6953.26	3476.63	2
Australia	Cloverd...	2005	NULL	17306.51	3461.302	5

Figure 3. In SQL Server, it's easy to compute aggregates for subgroups.

The order of the fields in the ROLLUP clause matters. The last one listed is summarized first. In Figure 3, you can see that the first level of summary is the whole year for a given city and country, because the month column is listed last. If you change the order in the ROLLUP clause to put the city last, as in Listing 6, the first summary level is a single month (and year), across all cities in a country; Figure 4 shows partial results.

Listing 6. The order of the fields in the ROLLUP clause matters. Changing the order changes what summaries you get.

```
GROUP BY ROLLUP(CountryRegion.Name,
                YEAR(OrderDate),
                MONTH(OrderDate),
                Address.City)
```

Name	City	nYear	nMonth	TotalSales	AvgSale	NumSales
Australia	Townsville	2008	7	66.11	13.222	5
Australia	Warrnambool	2008	7	660.20	73.3555	9
Australia	Wollongong	2008	7	367.84	91.96	4
Australia	NULL	2008	7	23555.63	63.1518	373
Australia	NULL	2008	NULL	6786807.30	1004.4113	6757
Australia	NULL	NU...	NULL	16322659...	1223.1292	13345
Canada	Haney	2005	7	3578.27	3578.27	1
Canada	Metchosin	2005	7	3578.27	3578.27	1
Canada	N. Vancouver	2005	7	3578.27	3578.27	1
Canada	Newton	2005	7	3374.99	3374.99	1
Canada	Royal Oak	2005	7	3578.27	3578.27	1
Canada	Shawnee	2005	7	4277.3682	2138.6841	2
Canada	NULL	2005	7	21965.4382	3137.9197	7
Canada	Burnaby	2005	8	3578.27	3578.27	1

Figure 4. When you change the order of fields in the ROLLUP clause, you get a different set of summaries.

The ROLLUP clause doesn't have to surround all the fields in the GROUP BY, only the ones for which you want summaries. So, if you don't need a grand total in the previous example, you can put CountryRegion.Name before the ROLLUP clause, as in Listing 7. Similarly, if you want summaries only for each city and year, put both CountryRegion.Name and Address.City before the ROLLUP clause. You can also put fields after the ROLLUP clause, but in my testing, the results aren't terribly useful.

Listing 7. Not all fields have to be included in ROLLUP, just those that should be summarized. With this GROUP BY clause, the results won't include grand totals because we're not rolling up the country.

```
GROUP BY CountryRegion.Name,
         ROLLUP(Address.City,
                YEAR(OrderDate),
                MONTH(OrderDate))
```

Name	City	nYear	nMonth	TotalSales	AvgSale	NumSales
United King...	NULL	2008	6	777041.13	1137.6883	683
United King...	NULL	2008	7	11942.84	60.6235	197
United King...	NULL	2008	NULL	3403936.78	951.0859	3579
United States	NULL	2008	1	1241090.59	783.0224	1585
United States	NULL	2008	2	1417644.54	857.6192	1653
United States	NULL	2008	3	1407198.05	831.6773	1692
United States	NULL	2008	4	1457717.98	807.1528	1806
United States	NULL	2008	5	2066050.29	979.1707	2110
United States	NULL	2008	6	1926201.99	965.031	1996
United States	NULL	2008	7	52011.40	63.2742	822
United States	NULL	2008	NULL	9567914.84	820.2944	11664
NULL	NULL	2008	NULL	27419405...	848.9505	32298
NULL	NULL	2006	9	350466.9...	1770.0353	198
NULL	NULL	2007	4	507965.2...	1716.0988	296

Figure 5. Using the GROUP BY clause in Listing 8 with the earlier query provides summaries for not just each city by year, each city overall, and each country, but also for each country by month and by year, and for each month and each year.

ROLLUP with cross-products

You can use two ROLLUP clauses in the same GROUP BY. Doing so gives you the cross-product of the two groups. That is, you get the results you'd get from either ROLLUP, but you also get combinations of the two.

For example, if you change the GROUP BY clause in Listing 5 to the one shown in Listing 8, you get all the rows you had before, but you also get summaries for each country for each month and year, as well as overall summaries for each month and for each year. Figure 5 shows part of the results. The complete query is included in this month's downloads as SalesByCountryCityRollupXProd.SQL.

Listing 8. You can use two ROLLUP clauses to generate the cross-product of the two sets of fields.

```
GROUP BY ROLLUP(YEAR(OrderDate),
                MONTH(OrderDate)),
         ROLLUP(CountryRegion.Name,
                Address.City)
```

As with a single ROLLUP clause, the order in which you list the ROLLUP clauses and the order of the fields within them determines both what summaries you get and the order of the records in the result (if you don't use an ORDER BY clause).

Adding descriptions to summaries

In all the examples so far, the null value indicates which field is being summarized. But you can put descriptive data in those fields instead.

Wrap the columns being rolled up with ISNULL() and specify the string you want in the summary rows as the alternate. (ISNULL() in SQL Server behaves like VFP's NVL() function, returning the first parameter unless it's null, in which case it returns the second parameter.) Listing 9 (SalesByCountryCityRollupWDesc.SQL in this month's downloads) shows the same query as Listing 5, except that each of the non-aggregated fields includes a description to use when it's summarized. Doing so requires changing

the year and month columns to character, of course. Figure 6 shows a chunk of the results.

Listing 9. Rather than having null indicate a summary row, use the description you want.

```
SELECT ISNULL(Person.CountryRegion.Name,
              'All countries') AS Country,
       ISNULL(Person.Address.City,
              'All cities') AS City,
       ISNULL(STR(YEAR(OrderDate)),
              'All years') AS OrderYear,
       ISNULL(STR(MONTH(OrderDate)),
              'All months') AS OrderMonth,
       SUM(SubTotal) AS TotalSales,
       AVG(SubTotal) AS AvgSale,
       COUNT(SubTotal) AS NumSales
  FROM Sales.Customer
    JOIN Person.Person
      ON Customer.PersonID =
         Person.BusinessEntityID
    JOIN Person.BusinessEntityAddress
      ON Person.BusinessEntityID =
        BusinessEntityAddress.BusinessEntityID
    JOIN Person.Address
      ON BusinessEntityAddress.AddressID =
         Address.AddressID
    JOIN Person.StateProvince
      ON Address.StateProvinceID =
         StateProvince.StateProvinceID
    JOIN Person.CountryRegion
      ON StateProvince.CountryRegionCode =
         CountryRegion.CountryRegionCode
    JOIN Sales.SalesOrderHeader
      ON Customer.CustomerID =
         SalesOrderHeader.CustomerID
    JOIN Sales.SalesOrderDetail
      ON SalesOrderHeader.SalesOrderID =
         SalesOrderDetail.SalesOrderID
GROUP BY ROLLUP(CountryRegion.Name,
                Address.City,
                YEAR(OrderDate),
                MONTH(OrderDate))
```

Country	City	OrderY...	OrderMo...	TotalSales	AvgSale	NumSales
Australia	Wollongong	2008	2	29543.26	1477.163	20
Australia	Wollongong	2008	3	46330.36	1494.5277	31
Australia	Wollongong	2008	4	42357.88	1033.119	41
Australia	Wollongong	2008	5	39493.51	1128.386	35
Australia	Wollongong	2008	6	64000.30	1280.006	50
Australia	Wollongong	2008	7	367.84	91.96	4
Australia	Wollongong	2008	All months	268554.13	1272.7683	211
Australia	Wollongong	All years	All months	618257.4...	1504.2761	411
Australia	All cities	All years	All months	16322659...	1223.1292	13345
Canada	Burnaby	2005	8	3578.27	3578.27	1
Canada	Burnaby	2005	All months	3578.27	3578.27	1
Canada	Burnaby	2006	3	3578.27	3578.27	1
Canada	Burnaby	2006	5	3578.27	3578.27	1

Figure 6. Including descriptions instead of null makes it easier to understand the summary lines.

More to come

In my next article, I'll look at additional ways to summarize results in SQL Server using the CUBE and GROUPING SETS clauses.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro and Taming Visual FoxPro's SQL. Her latest collaboration is VFPX: Open Source Treasure for the VFP Developer, available at www.foxrockx.com. Her other books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.